PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 9/46

A1

(11) International Publication Number: WO 99/44129

(43) International Publication Date: 2 September 1999 (02.09.99)

(21) International Application Number: PCT/US99/03399

(22) International Filing Date: 17 February 1999 (17.02.99)

(30) Priority Data:

60/076,048	26 February 1998 (26.02.98)	US
09/044,834	20 March 1998 (20.03.98)	US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, MS UPAL01-521, Palo Alto, CA 94303 (US).

(72) Inventors: WALDO, James, H.; 155 Ruby Road, Dracut, MA 01826 (US). WOLLRATH, Ann, M.; 9 Northwoods Road, Groton, MA 01450 (US). SCHEIFLER, Robert; 96 North Street, Somerville, MA 02144 (US). ARNOLD, Kenneth, C., R., C.,; 7 Moon Hill Road, Lexington, MA 02173 (US).

(74) Agents: GARRETT, Arthur, S.; Finnegan, Henderson, Farrabow, Garrett & Dunner, L.L.P., 1300 I Street, N.W., Washington, DC 20005-3315 (US) et al.

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD, APPARATUS, AND PRODUCT FOR LEASING OF GROUP MEMBERSHIP IN A DISTRIBUTED SYSTEM

(57) Abstract

A system for leasing a group membership in a distributed processing system is provided. In accordance with this system, a remote object requests from an activation group a membership in the activation group for a period of time. Responsive to this request, the activation group determines an appropriate lease period during which time the remote object becomes a member of the activation group and runs in the same address space as other objects of the activation group.

BNSDOCID: <WO___9944129A1_I_>

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

BNSDOCID: <WO___9944129A1_I_>

METHOD, APPARATUS, AND PRODUCT FOR LEASING OF GROUP MEMBERSHIP IN A DISTRIBUTED SYSTEM

RELATED APPLICATIONS

This is a continuation-in-part of U.S. Patent Application No. 08/729,421, filed on October 11, 1996, which is incorporated herein by reference.

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application.

Provisional U.S. Patent Application No. 60/076,048, entitled "Distributed Computing System," filed on February 26, 1998.

U.S. Patent Application No. 09/044,923, entitled "Method and System for Leasing Storage," bearing attorney docket no. 06502.0011-01000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,838, entitled "Method, Apparatus and Product for Leasing of Delegation Certificates in a Distributed System," bearing attorney docket no. 06502.0011-02000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,916, entitled "Leasing for Failure Detection," bearing attorney docket no. 06502.0011-04000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,933, entitled "Method for Transporting Behavior in Event Based System," bearing attorney docket no. 06502.0054-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,919, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," bearing attorney docket no. 06502.0062-01000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,938, entitled "Methods and Apparatus for Remote Method Invocation," bearing attorney docket no. 06502.0102-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/045,652, entitled "Method and System for Deterministic Hashes to Identify Remote Methods," bearing attorney docket no. 06502.0103-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,790, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," bearing attorney docket no.

BNSDOCID: <WO___9944129A1_I_>

06502.0104-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,930, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," bearing attorney docket no. 06502.0105-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,917, entitled "Suspension and Continuation of Remote Methods," bearing attorney docket no. 06502.0106-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,835, entitled "Method and System for Multi-Entry and Multi-Template Matching in a Database," bearing attorney docket no. 06502.0107-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,839, entitled "Method and System for In-Place Modifications in a Database," bearing attorney docket no. 06502.0108, and filed on the same date herewith.

U.S. Patent Application No. 09/044,945, entitled "Method and System for Typesafe Attribute Matching in a Database," bearing attorney docket no. 06502.0109-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,931, entitled "Dynamic Lookup Service in a Distributed System," bearing attorney docket no. 06502.0110-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," bearing attorney docket no. 06502.0112-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," bearing attorney docket no. 06502.0113-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,932, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," bearing attorney docket no. 06502.0114-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on February 26, 1998.

U.S. Patent Application No. 09/044,936, entitled "An Interactive Design Tool for

BNSDOCID: <WO___9944129A1_I_>

Persistent Shared Memory Spaces," bearing attorney docket no. 06502.0116-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,934, entitled "Polymorphic Token-Based Control," bearing attorney docket no. 06502.0117-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044.915, entitled "Stack-Based Access Control," bearing attorney docket no. 06502.0118-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,944, entitled "Stack-Based Security Requirements," bearing attorney docket no. 06502.0119-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,837, entitled "Per-Method Designation of Security Requirements," bearing attorney docket no. 06502.0120-00000, and filed on the same date herewith.

A. Field of the Invention

This invention relates generally to data processing systems and, more particularly, to leasing group memberships for objects in data processing systems.

B. Description of the Related Art

Proper resource management is an important aspect to efficient and effective use of computers. In general, resource management involves allocating resources (e.g., memory) in response to requests as well as deallocating resources at appropriate times, for example, when the requesters no longer require the resources. In general, the resources contain data referenced by computational entities (e.g., applications, programs, applets, etc.) executing in the computers.

In practice, when applications executing on computers seek to refer to resources, the computers must first allocate or designate resources so that the applications can properly refer to them. When the applications no longer refer to a resource, the computers can deallocate or reclaim the resource for reuse. In computers each resource has a unique "handle" by which the resource can be referenced. The handle may be implemented in various ways, such as an address, array index, unique value, pointer, etc.

Resource management is relatively simple for a single computer because the events indicating when resources can be reclaimed, such as when applications no longer refer to them or after a power failure, are easy to determine. Resource management for distributed systems

BNSDOCID: <WO__9944129A1_I_>

connecting multiple computers is more difficult because applications in several different computers may be using the same resource.

Disconnects in distributed systems can lead to the improper and premature reclamation of resources or to the failure to reclaim resources. For example, multiple applications operating on different computers in a distributed system may refer to resources located on other machines. If connections between the computers on which resources are located and the applications referring to those resources are interrupted, then the computers may reclaim the resources prematurely. Alternatively, the computers may maintain the resources in perpetuity, despite the extended period of time that applications failed to access the resources.

These difficulties have led to the development of systems to manage network resources, one of which is known as "distributed garbage collection." That term describes a facility provided by a language or runtime system for distributed systems that automatically manages resources used by an application or group of applications running on different computers in a network.

In general, garbage collection uses the notion that resources can be freed for future use when they are no longer referenced by any part of an application. Distributed garbage collection extends this notion to the realm of distributed computing, reclaiming resources when no application on any computer refers to them.

Distributed garbage collection must maintain integrity between allocated resources and the references to those resources. In other words, the system must not be permitted to deallocate or free a resource when an application running on any computer in the network continues to refer to that resource. This reference-to-resource binding, referred to as "referential integrity," does not guarantee that the reference will always grant access to the resource to which it refers. For example, network failures can make such access impossible. The integrity, however, guarantees that if the reference can be used to gain access to any resource, it will be the same resource to which the reference was first given.

Distributed systems using garbage collection must also reclaim resources no longer being referenced at some time in the finite future. In other words, the system must provide a guarantee against "memory leaks." A memory leak can occur when all applications drop references to a resource, but the system fails to reclaim the resource for reuse because, for example, of an incorrect determination that some application still refers to the resource.

BNSDOCID: <WO 9944129A1 I_>

Referential integrity failures and memory leaks often result from disconnections between applications referencing the resources and the garbage collection system managing the allocation and deallocation of those resources. For example, a disconnection in a network connection between an application referring to a resource and a garbage collection system managing that resource may prevent the garbage collection system from determining whether and when to reclaim the resource. Alternatively, the garbage collection system might mistakenly determine that, since an application has not accessed a resource within a predetermined time, it may collect that resource. A number of techniques have been used to improve the distributed garbage collection mechanism by attempting to ensure that such mechanisms maintain referential integrity without memory leaks. One conventional approach uses a form of reference counting, in which a count is maintained of the number of applications referring to each resource. When a resource's count goes to zero, the garbage collection system may reclaim the resource. Such a reference counting scheme only works, however, if the resource is created with a corresponding reference counter. The garbage collection system in this case increments the resource's reference count as additional applications refer to the resource, and decrements the count when an application no longer refers to the resource.

Reference counting schemes, however, especially encounter problems in the face of failures that can occur in distributed systems. Such failures can take the form of a computer or application failure or network failure that prevent the delivery of messages notifying the garbage collection system that a resource is no longer being referenced. If messages go undelivered because of a network disconnect, the garbage collection system does not know when to reclaim the resource.

To prevent such failures, some conventional reference counting schemes include "keep-alive" messages, which are also referred to as "ping back." According to this scheme, applications in the network send messages to the garbage collection system overseeing resources and indicate that the applications can still communicate. These messages prevent the garbage collection system from dropping references to resources. Failure to receive such a "keep-alive" message indicates that the garbage collection system can decrement the reference count for a resource and, thus, when the count reaches zero, the garbage collection system may reclaim the resource. This, however, can still result in the premature reclamation of resources following reference counts reaching zero from a failure to receive "keep-alive" messages because of

BNSDOCID: <WO___9944129A1_I_>

network failures. This violates the referential integrity requirement.

Another proposed method for resolving referential integrity problems in garbage collection systems is to maintain not only a reference count but also an identifier corresponding to each computational entity referring to a resource. See A. Birrell, et al., "Distributed Garbage Collection for Network Objects," No. 116, digital Systems Research Center, December 15, 1993. This method suffers from the same problems as the reference counting schemes. Further, this method requires the addition of unique identifiers for each computational entity referring to each resource, adding overhead that would unnecessarily increase communication within distributed systems and add storage requirements (i.e., the list of identifiers corresponding to applications referring to each resource).

SUMMARY OF THE INVENTION

In accordance with the present invention, referential integrity is guaranteed without costly memory leaks by leasing resources for a period of time during which the parties in a distributed system, for example, an application holding a reference to a resource and the garbage collection system managing that resource, agree that the resource and a reference to that resource will be guaranteed. At the end of the lease period, the guarantee that the reference to the resource will continue lapses, allowing the garbage collection system to reclaim the resource. Because the application holding the reference to the resource and the garbage collection system managing the resource agree to a finite guaranteed lease period, both can know when the lease and, therefore, the guarantee, expires. This guarantees referential integrity for the duration of a reference lease and avoids the concern of failing to free the resource because of network errors.

Consistent with an alternative embodiment of the present invention, as embodied and broadly described herein, a method for leasing group membership is provided. This method comprises the steps of receiving a lease request from a remote object specifying an activation group and a lease period and determining a lease period during which the remote object is associated with and runs in the same address space as the other objects, if any, in the activation group.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this

BNSDOCID: <WO 9944129A1 I >

specification, illustrate an embodiment of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings,

FIG. 1 is a flow diagram of the steps performed by the application call processor according to an implementation of the present invention;

FIG. 2 is a flow diagram of the steps performed by the server call processor to process dirty calls according to the implementation of the present invention;

FIG. 3 is a flow diagram of the steps performed by the server call processor to process clean calls according to the implementation of the present invention;

FIG. 4 is a flow diagram of the steps performed by the server call processor to initiate a garbage collection process according to the implementation of the present invention.

FIG. 5 is a diagram of a preferred flow of calls within a distributed processing system;

FIG. 6 is a block diagram of the components of the implementation of a method invocation service according to the present invention;

FIG. 7 is a diagram of a distributed processing system that can be used in an implementation of the present invention;

FIG. 8 is a diagram of the individual software components in the platforms of the distributed processing system according to the implementation of the present invention;

FIG. 9 is a diagram of a data processing system suitable for use by an alternative embodiment of the present invention;

FIG. 10 is a flow chart of the steps performed by an object when changing activation groups consistent with an alternative embodiment of the present invention;

FIG. 11 is a flow chart of the steps performed by the activation daemon of Figure 9; and

FIG. 12 is a flow chart of the steps performed by an activation group when a remote object requests membership into the activation group consistent with an alternative embodiment of the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to an implementation of the present invention as illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings and the following description to refer to the same or like parts.

The present invention may be implemented by computers organized in a conventional

BNSDOCID: <WO___9944129A1_I_>

distributed processing system architecture. The architecture for and procedures to implement this invention, however, are not conventional, because they provide a distributed garbage collection scheme that ensures referential integrity and eliminates memory leaks.

A. Overview

A method invocation (MI) component located in each of the computers in the distributed processing system implements the distributed garbage collection scheme of this invention. The MI component may consist of a number of software modules preferably written in the JAVA™ programming language.

In general, whenever an application in the distributed processing system obtains a reference to a distributed resource, by a name lookup, as a return value to some other call, or another method, and seeks to access the resource, the application makes a call to the resource or to an MI component managing the resource. That MI component, called a managing MI component, keeps track of the number of outstanding references to the resource. When the number of references to a reference is zero, the managing MI component can reclaim the resource. The count of the number of references to a resource is generally called the "reference count" and the call that increments the reference count may be referred to as a "dirty call."

When an application no longer requires a distributed resource, it sends a different call to the resource or the managing MI component. Upon receipt of this call, the managing MI component decrements the reference count for the resource. This call to drop a reference may be referred to as a "clean call."

In accordance with an implementation of the present invention, a dirty call can include a requested time interval, called a lease period, for the reference to the resource. Upon receipt of the dirty call, the managing MI component sends a return call indicating a period for which the lease was granted. The managing MI component thus tracks the lease period for those references as well as the number of outstanding references. Consequently, when the reference count for a resource goes to zero or when the lease period for the resource expires, the managing MI component can reclaim the resource.

B. Procedure

An application call processor in an MI component performs the steps of the application call procedure 100 illustrated in FIG. 1. The server call processor in the managing MI component performs the steps of the procedures 200, 300, and 400 illustrated in FIGs. 2-4,

BNSDOCID: <WO_ 9944129A1 I >

respectively. The managing MI component's garbage collector performs conventional procedures to reclaim resources previously bound to references in accordance with instructions from the server call processor. Accordingly, the conventional procedures of the garbage collector will not be explained.

1. Application Call Processor

FIG. 1 is a flow diagram of the procedure 100 that the application call processor of the MI component uses to handle application requests for references to resources managed by the same or another MI component located in the distributed processing system.

After an application has obtained a reference to a resource, the application call processor sends a dirty call, including the resource's reference and a requested lease period to the managing MI component for the resource (step 110). The dirty call may be directed to the resource itself or to the managing MI component.

The application call processor then waits for and receives a return call from the managing MI component (step 120). The return call includes a granted lease period during which the managing MI component guarantees that the reference of the dirty call will be bound to its resource. In other words, the managing MI component agrees not to collect the resource corresponding to the reference of a dirty call for the grant period. If the managing MI component does not provide a grant period, or rejects the request for a lease, then the application call processor will have to send another dirty call until it receives a grant period.

The application call processor monitors the application's use of the reference and, either when the application explicitly informs the application call processor that the reference is no longer required or when the application call processor makes this determination on its own (step 130), the application call processor sends a clean call to the managing MI component (step 140). In a manner similar to the method used for dirty calls, the clean call may be directed to the referenced resource and the managing MI component will process the clean call. Subsequently, the application call processor eliminates the reference from a list of references being used by the application (step 150).

If the application is not yet done with the reference (step 130), but the application call processor determines that the grant period for the reference is about to expire (step 160), then the application call processor repeats steps 110 and 120 to ensure that the reference to the resource is maintained by the managing MI component on behalf of the application.

BNSDOCID: <WO___9944129A1_I_>

2. Server Call Processor

The MI component's server call processor performs three main procedures: (1) handling dirty calls; (2) handling incoming clean calls; and (3) initiating a garbage collection cycle to reclaim resources at the appropriate time.

(i) Dirty Calls

FIG. 2 is a flow diagram of the procedure 200 that the MI component's server call processor uses to handle requests to reference resources, i.e., dirty calls, that the MI software component manages. These requests come from application call processors of MI components in the distributed processing system, including the application call processor of the same MI component as the server call processor handling requests.

First, the server call processor receives a dirty call (step 210). The server call processor then determines an acceptable grant period (step 220). The grant period may be the same as the requested lease period or some other time period. The server call processor determines the appropriate grant period based on a number of conditions including the amount of resource required and the number of other grant periods previously granted for the same resource.

When the server call processor determines that a resource has not yet been allocated for the reference of a dirty call (step 230), the server call processor allocates the required resource (step 240).

The server call processor then increments a reference count corresponding to the reference of a dirty call (step 250), sets the acceptable grant period for the reference-to-resource binding (step 260), and sends a return call to an application call processor with the grant period (step 270). In this way, the server call processor controls incoming dirty calls regarding references to resources under its control.

Applications can extend leases by sending dirty calls with an extension request before current leases expire. As shown in procedure 200, a request to extend a lease is treated just like an initial request for a lease. An extension simply means that the resource will not be reclaimed for some additional interval of time, unless the reference count goes to zero.

(ii) Clean Calls

The MI component's server call processor also handles incoming clean calls from application call processors. When an application in the distributed processing system no longer requires a reference to a resource, it informs the MI component managing the resource for that

BNSDOCID: <WO 9944129A1_I_>

reference so that the resource may be reclaimed for reuse. Fig. 3 is a flow diagram of the procedure 300 with the steps that the MI component's server call processor uses to handle clean calls.

When the server call processor receives a clean call with a reference to a resource that the MI component manages (step 310), the server call processor decrements a corresponding reference count (step 320). The clean call may be sent to the resource, with the server call processor monitoring the resource and executing the procedure 300 to process the call. Subsequently, the server call processor sends a return call to the MI component that sent the clean call to acknowledge receipt (step 330). In accordance with this implementation of the present invention, a clean call to drop a reference may not be refused, but it must be acknowledged.

(iii) Garbage Collection

The server call processor also initiates a garbage collection cycle to reclaim resources for which it determines that either no more references are being made to the resource or that the agreed lease period for the resource has expired. The procedure 400 shown in FIG. 4 includes a flow diagram of the steps that the server call processor uses to initiate a garbage collection cycle.

The server call processor monitors reference counts and granted lease periods and determines whether a reference count is zero for a resource managed by the MI component, or the grant period for a reference has expired (step 410). When either condition exists, the server call processor initiates garbage collection (step 420) of that resource. Otherwise, the server call processor continues monitoring the reference counts and granted lease periods.

C. Call Flow

FIG. 5 is a diagram illustrating the flow of calls among MI components within the distributed processing system. Managing MI component 525 manages the resources 530 by monitoring the references to those resources 530 (see garbage collect 505). Because the managing MI components 525 manages the resources, the server call processor of managing MI component 525 performs the operations of this call flow description.

FIG. 5 also shows that applications 510 and 540 have corresponding MI components 515 and 545, respectively. Each of the applications 510 and 540 obtains a reference to one of the resources 530 and seeks to obtain access to one of the resources 530 such that a reference is bound to the corresponding resource. To obtain access, applications 510 and 540 invoke their

BNSDOCID: <WO___9944129A1_I_>

corresponding MI components 515 and 545, respectively, to send dirty calls 551 and 571, respectively, to the MI component 525. Because the MI components 515 and 525 handle application requests for access to resources 530 managed by another MI component, such as managing MI component 525, the application call processors of MI components 515 and 545 perform the operations of this call flow description.

In response to the dirty calls 551 and 571, managing MI component 525 sends return calls 552 and 572, respectively, to each of the MI components 515 and 545, respectively. The dirty calls include granted lease periods for the references of the dirty calls 551 and 571.

Similarly, FIG. 5 also shows MI components 515 and 545 sending clean calls 561 and 581, respectively, to managing MI component 525. Clean calls 561 and 581 inform managing MI component 525 that applications 510 and 540, respectively, no longer require access to the resource specified in the clean calls 561 and 581. Managing MI component 525 responds to clean calls 561 and 581 with return calls 562 and 582, respectively. Return calls 562 and 582 differ from return calls 552 and 572 in that return calls 562 and 582 are simply acknowledgments from MI component 525 of the received clean calls 561 and 581.

Both applications 510 and 540 may request access to the same resource. For example, application 510 may request access to "RESOURCE(1)" while application 540 was previously granted access to that resource. MI component 525 handles this situation by making the resource available to both applications 510 and 540 for agreed lease periods. Thus, MI component 525 will not initiate a garbage collection cycle to reclaim the "RESOURCE(1)" until either applications 510 and 540 have both dropped their references to that resource or the latest agreed periods has expired, whichever event occurs first.

By permitting more than one application to access the same resource simultaneously, the present invention also permits an application to access a resource after it sent a clean call to the managing MI component dropping the reference to the resource. This occurs because the resource is still referenced by another application or the reference's lease has not yet expired so the managing MI component 525 has not yet reclaimed the resource. The resource, however, will be reclaimed after a finite period, either when no more applications have leases or when the last lease expires.

BNSDOCID: <WO 9944129A1 I >

D. MI Components

FIG. 6 is a block diagram of the modules of an MI component 600 according to an implementation of the present invention. MI component 600 can include a reference component 605 for each reference monitored, application call processor 640, server call processor 650, and garbage collector 660.

Reference component 605 preferably constitutes a table or comparable structure with reference data portions 610, reference count 620, and grant period register 630. MI component 600 uses the reference count 620 and grant period 630 for each reference specified in a corresponding reference data portion 610 to determine when to initiate garbage collector 660 to reclaim the corresponding resource.

Application call processor 640 is the software module that performs the steps of procedure 100 in FIG. 1. Server call processor 650 is the software module that performs the steps of procedures 200, 300, and 400 in FIGs. 2-4. Garbage collector 660 is the software module that reclaims resources in response to instructions from the server call processor 650, as explained above.

E. Distributed Processing System

FIG. 7 illustrates a distributed processing system 50 which can be used to implement the present invention. In FIG. 7, distributed processing system 50 contains three independent and heterogeneous platforms 100, 200, and 300 connected in a network configuration represented by the network cloud 55. The composition and protocol of the network configuration represented in FIG. 7 by the cloud 55 is not important as long as it allows for communication of the information between platforms 700, 800 and 900. In addition, the use of just three platforms is merely for illustration and does not limit the present invention to the use of a particular number of platforms. Further, the specific network architecture is not crucial to this invention. For example, another network architecture that could be used in accordance with this invention would employ one platform as a network controller to which all the other platforms would be connected.

In the implementation of distributed processing system 50, platforms 700, 800 and 900 each include a processor 710, 810, and 910 respectively, and a memory, 750, 850, and 950, respectively. Included within each processor 710, 810, and 910, are applications 720, 820, and 920, respectively, operating systems 740, 840, and 940, respectively, and MI components 730,

BNSDOCID: <WO___9944129A1_I_>

830, and 930, respectively.

Applications 720, 820, and 920 can be programs that are either previously written and modified to work with the present invention, or that are specially written to take advantage of the services offered by the present invention. Applications 720, 820, and 920 invoke operations to be performed in accordance with this invention.

MI components 730, 830, and 930 correspond to the MI component 600 discussed above with reference to FIG. 6.

Operating systems 740, 840, and 940 are standard operating systems tied to the corresponding processors 710, 810, and 910, respectively. The platforms 700, 800, and 900 can be heterogenous. For example, platform 700 has an UltraSparc® microprocessor manufactured by Sun Microsystems Corp. as processor 710 and uses a Solaris® operating system 740. Platform 800 has a MIPS microprocessor manufactured by Silicon Graphics Corp. as processor 810 and uses a Unix operating system 840. Finally, platform 900 has a Pentium microprocessor manufactured by Intel Corp. as processor 910 and uses a Microsoft Windows 95 operating system 940. The present invention is not so limited and could accommodate homogenous platforms as well.

Sun, Sun Microsystems, Solaris, Java, and the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UltraSparc and all other SPARC trademarks are used under license and are trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

Memories 750, 850, and 950 serve several functions, such as general storage for the associated platform. Another function is to store applications 720, 820, and 920, MI components 730, 830, and 930, and operating systems 740, 840, and 940 before execution by the respective processor 710, 810, and 910. In addition, portions of memories 750, 850, and 950 may constitute shared memory available to all of the platforms 700, 800, and 900 in network 50.

E. MI Services

The present invention may be implemented using a client/server model. The client generates requests, such as the dirty calls and clean calls, and the server responds to requests.

Each of the MI components 730, 830 and 930 shown in FIG. 7 preferably includes both client components and server components. FIG. 8, which is a block diagram of a client platform

BNSDOCID: <WO 9944129A1 I >

1000 and a server platform 1100, applies to any two of the platforms 700, 800, and 900 in FIG. 7.

Platforms 1000 and 1100 contain memories 1050 and 1150, respectively, and processors 1010 and 1110, respectively. The elements in the platforms 1000 and 1100 function in the same manner as similar elements described above with reference to FIG. 7. In this example, processor 1010 executes a client application 1020 and processor 1110 executes a server application 1120. Processors 1010 and 1110 also execute operating systems 1040 and 1140, respectively, and MI components 1030 and 1130, respectively.

MI components 1030 and 1130 each include a server call processor 1031 and 1131, respectively, an application call processor 1032 and 1132, respectively, and a garbage collector 1033 and 1133, respectively. Each of the MI components 1030 and 1130 also contains reference components, including reference data portions 1034 and 1134, respectively, reference counts 1035 and 1135, respectively, and grant period registers 1036 and 1136, respectively, for each reference that the respective MI component 1030 or 1130 monitors.

Application call processors 1032 and 1132 represent the client service and communicate with server call processors 1031 and 1131, respectively, which represent the server service. Because platforms 1000 and 1100 contain a server call processor, an application call processor, a garbage collector, and reference components, either platform can act as a client or a server.

For purposes of the discussion that follows, however, platform 1000 is designated the client platform and platform 1100 is designated as the server platform. In this example, client application 1020 obtains references to distributed resources and uses MI component 1030 to send dirty calls to the resources managed by MI component 1130 of server platform 1100.

Additionally, server platform 1100 may be executing a server application 1120. Server application 1120 may also use MI component 1130 to send dirty calls, which may be handled by MI component 1130 when the resources of those dirty calls are managed by MI component 1130. Alternatively, server application 1120 may use MI component 1130 to send dirty calls to resources managed by MI component 1030.

Accordingly, server call processor 1031, garbage collector 1033, and reference count 1035 for MI component 1030 of client platform 1000 are not active and are therefore presented in FIG. 8 as shaded. Likewise, application call processor 1132 of MI component 1130 of the server platform 1100 is shaded because it is also dormant.

BNSDOCID: <WO___9944129A1_I_>

When client application 1020 obtains a reference corresponding to a resource, application call processor 1032 sends a dirty call, which server call processor 1131 receives. The dirty call includes a requested lease period. Server call processor 1131 increments the reference count 1135 for the reference in the dirty call and determines a grant period. In response, server call processor 1131 sends a return call to application call processor 1030 with the grant period. Application call processor 1032 uses the grant period to update recorded grant period 1035, and to determine when the resource corresponding to the reference of its dirty call may be reclaimed.

Server call processor 1131 also monitors the reference counts and grant periods corresponding to references for resources that it manages. When one of its reference counts 1135 is zero, or when the grant period 1135 for a reference has expired, whichever event occurs first, server call processor 1131 may initiate the garbage collector 1133 to reclaim the resource corresponding to the reference that has a reference count of zero or an expired grant period.

The leased-reference scheme according to the implementation of the present invention does not require that the clocks on the platforms 1000 and 1100 involved in the protocol be synchronized. The scheme merely requires that they have comparable periods of increase. Leases do not expire at a particular time, but rather expire after a specific time interval. As long as there is approximate agreement on the interval, platforms 1000 and 1100 will have approximate agreement on the granted lease period. Further, since the timing for the lease is, in computer terms, fairly long, minor differences in clock rate will have little or no effect.

The transmission time of the dirty call can affect the protocol. If MI component 1030 holds a lease to reference and waits until just before the lease expires to request a renewal, the lease may expire before the MI component 1130 receives the request. If so, MI component 1130 may reclaim the resource before receiving the renewal request. Thus, when sending dirty calls, the sender should add a time factor to the requested lease period in consideration of transmission time to the platform handling the resource of a dirty call so that renewal dirty calls may be made before the lease period for the resource expires.

F. Conclusion

In accordance with the present invention a distributed garbage collection scheme ensures referential integrity and eliminates memory leaks by providing granted lease periods corresponding to references to resources in the distributed processing system such that when the granted lease periods expire, so do the references to the resources. The resources may then be

BNSDOCID: <WO 9944129A1 I >

collected. Resources may also be collected when they are no longer being referenced by processes in the distributed processing system with reference to counters assigned to the references for the resources.

Alternative Embodiment of the Present Invention

The leasing technique, described above, relates to garbage collection. However, an alternative embodiment of the present invention, as described below, can be used to govern object membership in an activation group.

In a distributed system that utilizes the Java™ programming environment ("the exemplary distributed system"), objects contain methods that may be invoked remotely. These methods can be invoked by a client using the Java™ remote method invocation system (RMI), sold as part of the Java Software development kit, available from Sun Microsystems, Inc. of Mountain View, CA. In the exemplary distributed system, when a client invokes a method on a remote object, the object may not already be in memory on the remote machine. In this case, a component of RMI, known as the activation daemon, on the remote machine starts a Java™ virtual machine (JVM) and then activates the object. The Java Virtual Machine is also provided as part of the Java software development kit and is described in greater detail in Lindholm and Yellin, The Java Virtual Machine Specification, Addison-Wesley (1997), which is incorporated herein by reference. "Activating an object" entails bringing the object into memory from secondary storage and invoking the requested method. It should be noted that both the object and the JVM run in a single address space. This process is repeated each time a method on a remote object not already in memory is called by the client. The exemplary distributed system is described in greater detail in copending U.S. Patent Application No. _____, entitled "Dynamic Lookup Service in a Distributed System," which has previously been incorporated by reference.

A drawback to bringing objects into their own address space one at a time is that it is inefficient. Since each object is loaded into a separate address space, a loaded object is required to cross process boundaries to access another loaded object, which requires a significant amount of processing time. Accordingly, it is beneficial to load related objects into the same address space to avoid crossing process boundaries and facilitating locality of reference, since related objects tend to access each other often. In the alternative embodiment, related objects are grouped into an object group or a set of objects. Since the objects in a group are related to each

other, it is efficient for them to run on the same JVM and, consequently, in the same address space. By running on the same JVM in the same address space, related objects can access each other more efficiently (i.e., directly, without having to cross process boundaries).

In the situation where related objects are grouped, when a client program invokes a method on a remote object, the activation daemon determines if the remote object is a member of a group of remote objects, known as an activation group. If the object belongs to an activation group and the group is not already loaded in memory, the activation daemon will start a JVM and load the object into the address space of the JVM. Subsequent invocations of methods on the other objects in the activation group will cause the activation daemon to load these objects into the same address space. Once the requested object is loaded on the JVM, the activation daemon will activate the object, and the requested method is invoked.

A drawback of grouping related objects together in the JVM is that a particular object could monopolize the activation group's available resources to the detriment of the other group members. For example, in a white-board program with multiple objects wanting to draw to a common area of the display using a cursor object, the cursor object is loaded into an activation group of objects who want to use it. Placing the cursor object in the same address space as the objects who want to use it allows the cursor to respond very quickly to a command because the cursor object is accessed without crossing process boundaries. The problem arises, however, when a particular object monopolizes the cursor object for so long as to negatively affect the other objects who want to use the cursor object.

The alternative embodiment eliminates this threat of monopolization by applying the leasing technique to the objects located in the activation group. Leasing ensures that no object in the activation group could monopolize another object. For instance, each object has to request a lease to join the activation group (i.e., to gain membership into the activation group). The activation group, itself, decides how long to grant the lease. For a white-board program, the activation group would grant very small leases. This ensures that no object could monopolize the cursor object, for when each object's lease expires, it is ejected from the activation group.

In the alternative embodiment, an object requests a lease to be a member of an activation group. The object must specify a period of time (the lease period) for which it desires membership, and the activation group, represented and managed by an object (the activation group object), decides how long to grant the lease. In addition to the period of time, the lease

BNSDOCID: <WO___9944129A1 I_>

request contains an indication of whether the object wants an exact lease. When requesting a lease, the object may request an exact lease period. In this case, the activation group only grants a lease to the object if the lease period granted would be the entire lease period requested.

Every object is governed by a lease into an activation group. Even if a lease expires, the object is provided with a membership into its own activation group until it requests to be a member of a different activation group.

Once an object requests a lease from an activation group, the activation group returns to the object a lease object, which includes methods for determining the duration of the lease, for renewing the lease, and for canceling the lease. The lease object is an instance of a class that may be extended in many ways to offer more functionality, but the basic class is defined as follows:

```
interface Lease {
        public long getDuration ( );
        public void cancel ( ) throws  UnknownLeaseException,
                                       RemoteException;
        public void renew (long renewDuration) throws
                                LeaseDeniedException,
                                UnknownLeaseException,
                                RemoteException;
        }
```

This class contains the getDuration method, the cancel method, and the renew method. Invoking the "getDuration" method provides an object with the length of the granted lease period. This period represents the most recent lease granted by the activation group. It is the object's responsibility, however, to determine the amount of time outstanding on the lease.

The "renew" method permits the remote object to renew the lease, asking for more time, without having to re-initiate the original lease request. Situations where an object may desire to renew the lease include when the original lease proves to be insufficient (i.e., the remote object requires additional membership), or when only a partial lease (i.e., less than the requested lease) was granted.

An object may use the renew method to request an additional lease period, or the object may continually invoke the renew method multiple times until many additional lease periods are granted. The renew method has no return value; if the renewal is granted, the new lease period will be reflected in the lease object on which the call was made. If the activation group is unable or unwilling to renew the lease, the reason is set forth in the lease object on which the call was

made.

An object invokes the "cancel" method when the object wishes to withdraw from the activation group, but there is still time left on the lease. The cancel method allows the activation group to remove the object from the activation group such that the object can no longer access any of the objects in the activation group without crossing process boundaries. In contrast, upon the end of a lease (i.e., natural termination), the activation group knows to eject the object. In this case, the object has no obligation to notify the activation group upon the natural termination of the lease.

Figure 9 depicts a data processing system 9000 suitable for use by an alternative embodiment of the present invention. The data processing system 9000 includes a computer system 9002 connected, via a network 9006, to a second computer system 9004. The network 9006 may be a local area network, a wide area network or the Internet.

The computer system 9002 includes a memory 9008, a secondary storage device 9010, a central processing unit (CPU) 9012, an input device 9016, and a video display 9014. The memory 9008 further includes a Java™ runtime system 9042, which includes a Java virtual machine (JVM) 9044, RMI 9046, and an activation group 9040. The activation group 9040 may have some objects 9048 associated with it which have not been activated and thus still reside on the secondary storage device 9010. In addition, the memory 9008 includes a Java runtime system 9022, which includes a JVM 9024, RMI 9026, and an activation group 9018, and a Java runtime system 9102, including an activation daemon 9100, a JVM 9104, and RMI 9106. The activation daemon 9100 is responsible for activating objects 9048 by bringing them into memory to be part of either activation group 9040 or activation group 9018.

The computer system 9004 includes a memory 9050, a secondary storage device 9052, a CPU 9054, an input device 9058, and a video display 9056. The memory 9050 further includes a Java runtime system 9062, which includes JVM 9064, RMI 9066, and a client program 9040 which can request invocation of a method of a remote object (e.g., one of objects 9048) via a call to RMI 9066. One skilled in the art will appreciate that data processing system 9000 and computers 9002 and 9004 may contain additional or different components.

Although aspects of the alternative embodiment are described as being stored in memory, one skilled in the art will appreciate that these aspects may also be stored on or read from other computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or

BNSDOCID: <WO 9944129A1 I >

CD-Rom; a carrier wave from the Internet, or other forms of RAM or ROM. Additionally, although the alternative embodiment is described as operating in a Java™ programming environment, one skilled in the art will appreciate that the alternative embodiment can operate in other environments as well.

Figure 10 depicts a flow chart of the steps performed by an object to become a member of a new activation group. For example, an object in activation group 9040 may want to become a member of activation group 9018. The object may want to do so because it plans on performing processing that may be related to activation group 9018 (e.g., graphics-related processing) and thus wishes to gain the benefit of locality of reference by running within this activation group having objects that perform graphics operations. To become a member of the new activation group, the object sends a lease request to the activation group 9018 (step 10004). In this step, the object sends a lease request to the activation group, which is an object, passing a number of parameters including a requested lease period.

After sending the lease request, the object determines whether the lease was successful by determining whether it received a lease object (step 10006). If the lease was successful, the object notifies the activation daemon 9100 of its new association with activation group 9018 (step 1008). When notifying the activation daemon, the object provides a copy of the lease object, which can be used by the activation daemon to determine the length of the lease period. After notifying the activation daemon, the object runs as part of the activation group 9108 (step 10010). In this step, if the object becomes activated, it will execute on the Java runtime system 9022. At some point later, the object determines whether it needs more time to run as part of the activation group (step 10012). If the object needs more time, the object invokes the renew method on the lease object returned via the lease request (step 10014). Then, the object determines if the renew request was successful (step 10016). If so, processing continues to step 10010. Otherwise, processing ends.

If the object does not need more time, the object invokes the cancel method on the lease object to terminate the lease (step 10018). This step cancels the lease and, consequently, the object's membership in the activation group. Next, the object obtains a lease to another activation group, perhaps the activation group 9040 from which it came (step 10020). It should be noted that if the object does not join another activation group, the next time it is activated, the activation daemon will make it a member of its own activation group.

BNSDOCID: <WO__9944129A1_I_>

Figure 11 depicts a flow chart of the steps performed by the activation daemon when a request is made to activate an object such as when client program 9060 invokes RMI 9066 to invoke a method on a remote object (e.g., one of the objects 9048). The first step performed by the activation daemon 9100 is to receive a request from RMI 9106 (step 11002). RMI 9106 may be invoked as a result of a remote method call from client 9060. When client 9060 invokes a remote method, it does so via RMI 9066. After RMI 9066 invokes a remote method, the stub used for the invocation contains the network port that the object used for communication as well as a reference to its activation daemon 9100. Thus, when receiving a request to invoke a remote method, RMI 9066 first attempts to invoke the method via its object's network port. If successful, the object has remained active and in memory since the last invocation of one of its methods via RMI 9066. However, if this fails, RMI 9066 sends a request to the activation daemon 9100 via RMI 9106, because the object needs to be activated.

After receiving a request from RMI 9106, the activation daemon determines if the lease for the object has expired (step 11004). In the alternative embodiment, each object is a member of an activation group and may usually have an outstanding lease. The activation daemon 9100 maintains a mapping of all objects in computer 9002 to the corresponding activation groups with which they are associated. If the activation daemon determines that the lease has expired, the activation daemon places the requested object into its own activation group (step 11006). In this step, the activation daemon stores an indication that the requested object is a member of its own activation group. After placing the object in its own activation group, the activation daemon starts a JVM and activates the object (step 11008). In this step, the activation daemon starts a JVM and loads the object into the JVM's address space from secondary storage 9010. Additionally, the activation daemon invokes the requested method on the object.

If the activation daemon determines that the lease has not expired, the activation daemon determines if the activation group in which the object is a member is currently running in a JVM (step 11010). If not, the activation daemon starts a JVM for the activation group (step 11012), and the activation daemon then activates the object (step 10014). In this step, the activation daemon brings the object from secondary storage into memory (the address space of the started JVM) and invokes the requested method.

Figure 12 depicts a flow chart of the steps performed by an activation group (e.g., 9018) when an object requests a lease. The first step performed by the activation group is to examine

BNSDOCID: <WO 9944129A1 >

the parameters of the lease request to verify the propriety of the request (step 12004). As stated above, the parameters include a requested lease period and an exact lease indication.

After examining the parameters, the activation group determines if the request is proper (step 12006). For example, the activation group verifies that a desired lease period is specified, and if an exact lease is requested, the activation group determines if it can grant the request. For example, the activation group may be unwilling to grant a lease for the period of time for which the object has requested an exact lease. If the activation group determines that the lease request is improper, the activation group returns an exception object indicating the error (step 12008) and processing ends.

If the request is proper, the activation group determines a sufficient lease period (step 12010). The activation group then creates a lease object and returns it to the remote object (step 12012).

The foregoing description of an implementation of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. The scope of the invention is defined by the claims and their equivalents.

WHAT IS CLAIMED IS:

1. A method in a processing system, comprising the steps of:

receiving a lease request from a remote object, the lease request specifying an activation group of objects and a requested lease period; and

determining a lease period during which the remote object becomes a member of the activation group.

BNSDOCID: <WO 9944129A1 I >

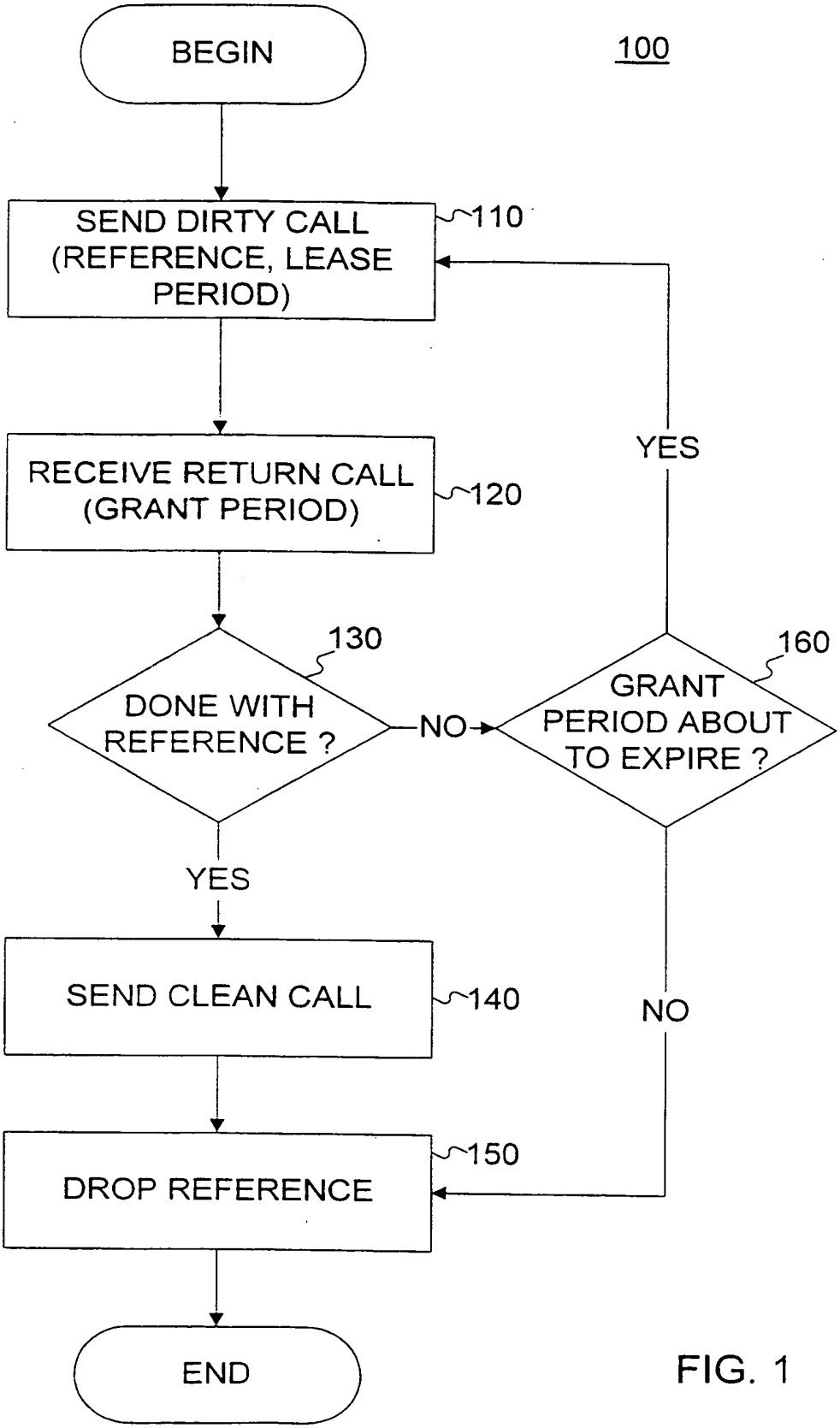


FIG. 1

BNSDOCID: <WO___9944129A1_I_>

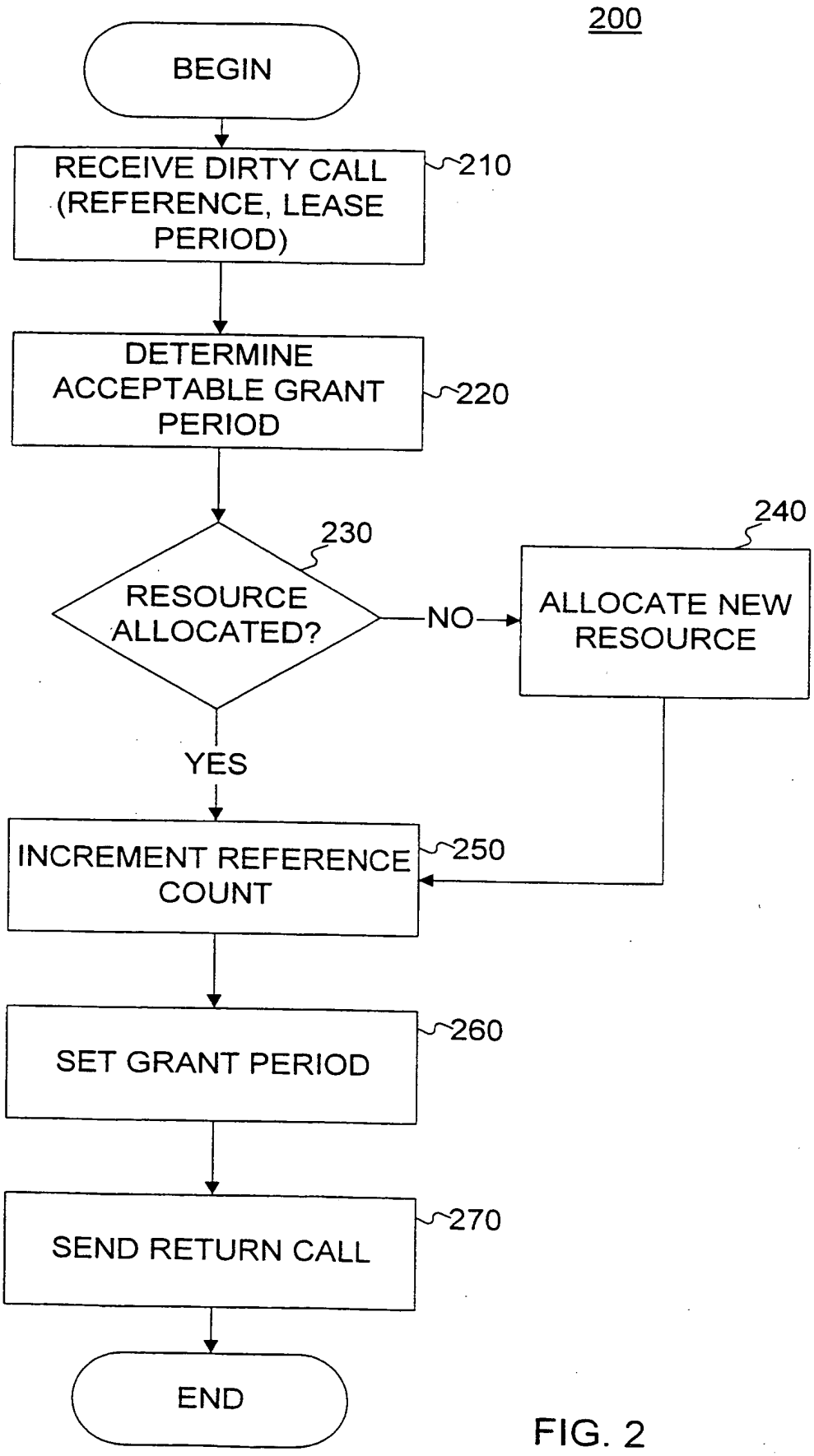


FIG. 2

BNSDOCID: <WO___9944129A1 | >

300

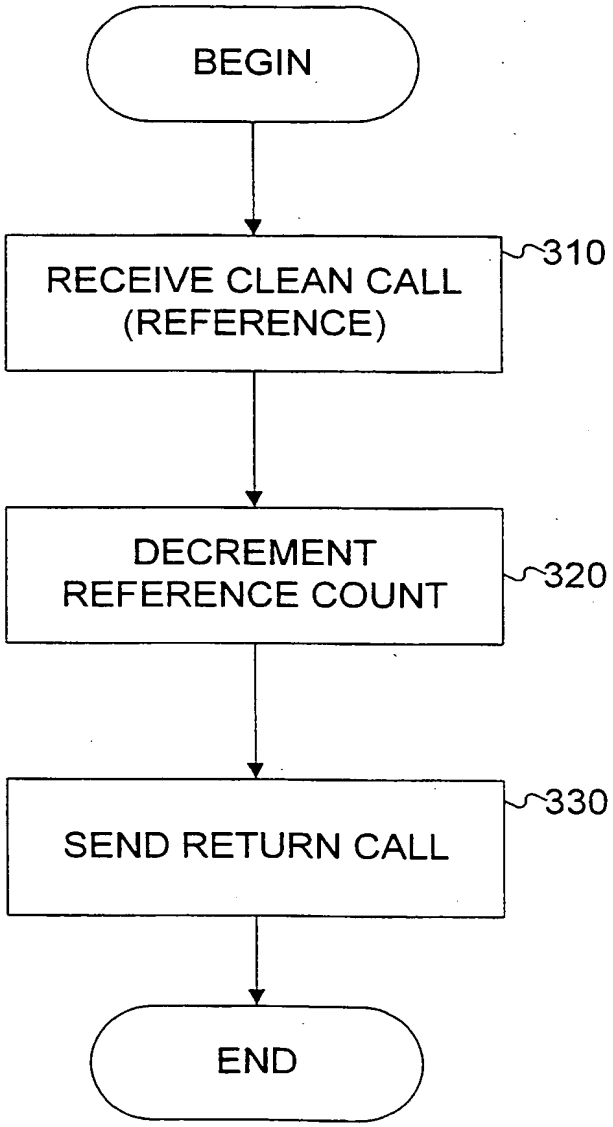


FIG. 3

BNSDOCID: <WO___9944129A1_I_>

400

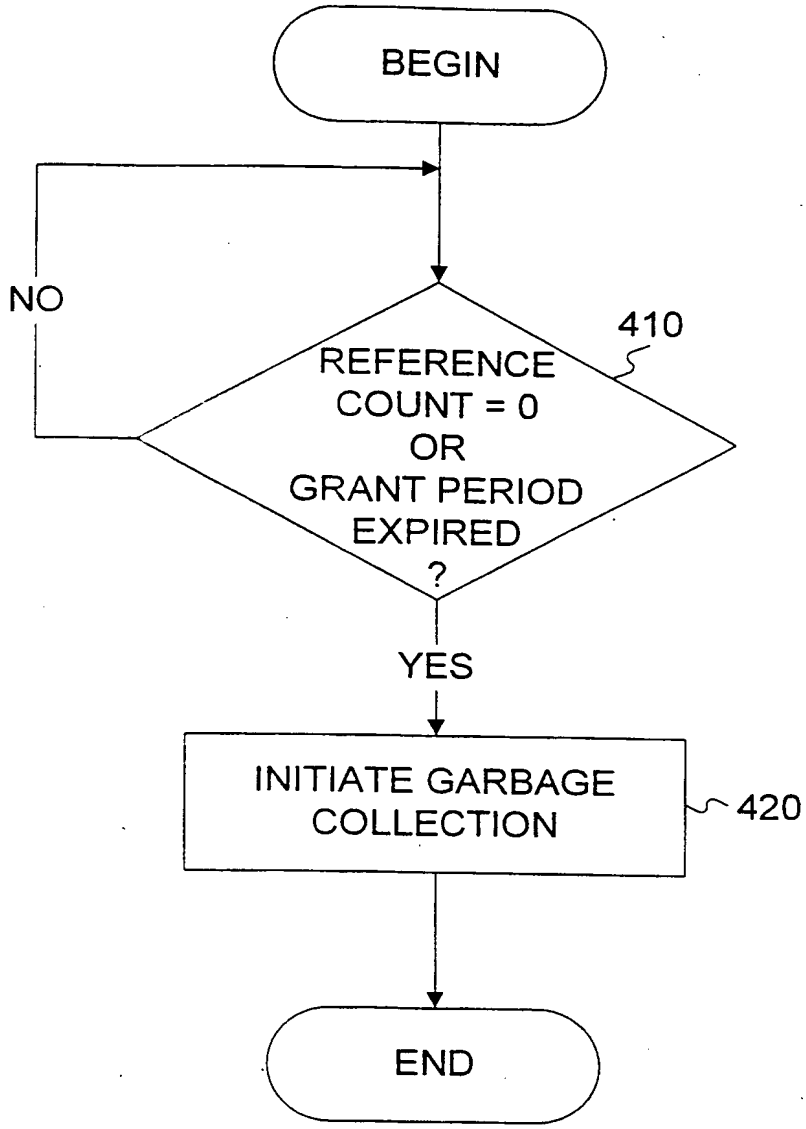
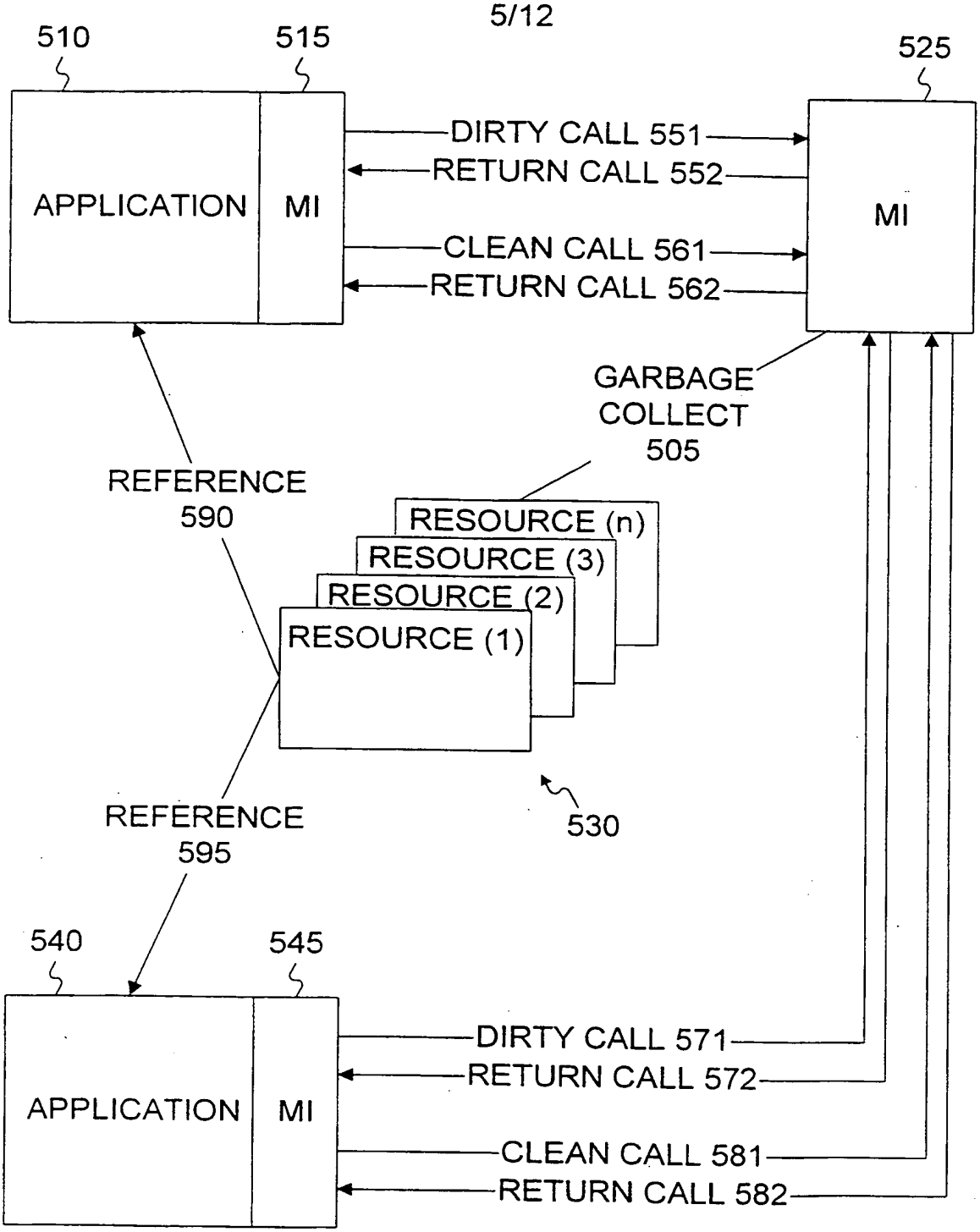


FIG. 4

BNSDOCID: <WO 9944129A1 I >

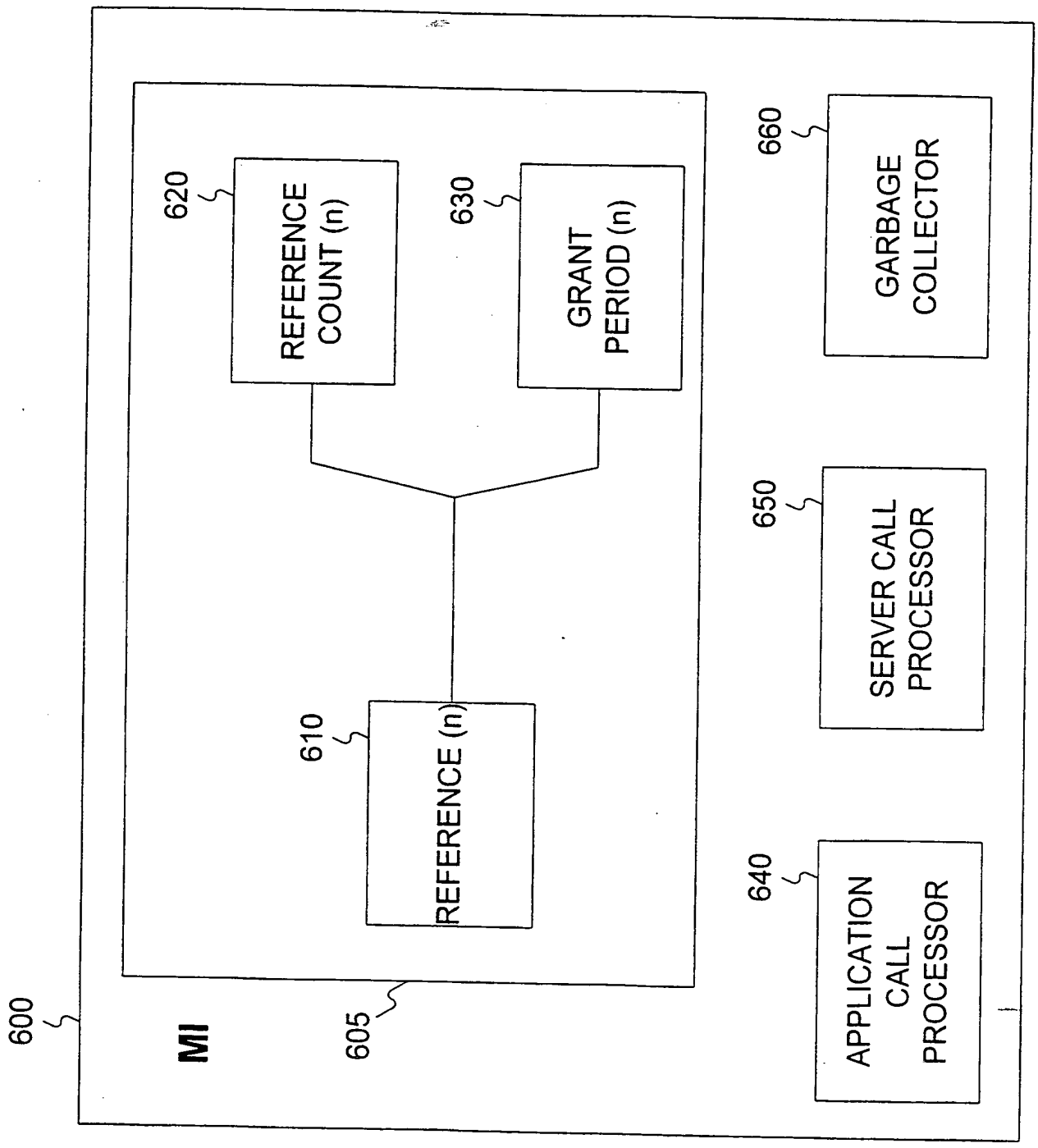


——DIRTY CALL——→	INCREMENT REFERENCE COUNTER
——CLEAN CALL——→	DECREMENT REFERENCE COUNTER
REFERENCE COUNTER = 0	INITIATE GARBAGE COLLECTOR
GRANT PERIOD EXPIRED	INITIATE GARBAGE COLLECTOR

FIG. 5

BNSDOCID: <WO___9944129A1_I_>

FIG. 6



BNSDOCID <WO___9944129A1 I >

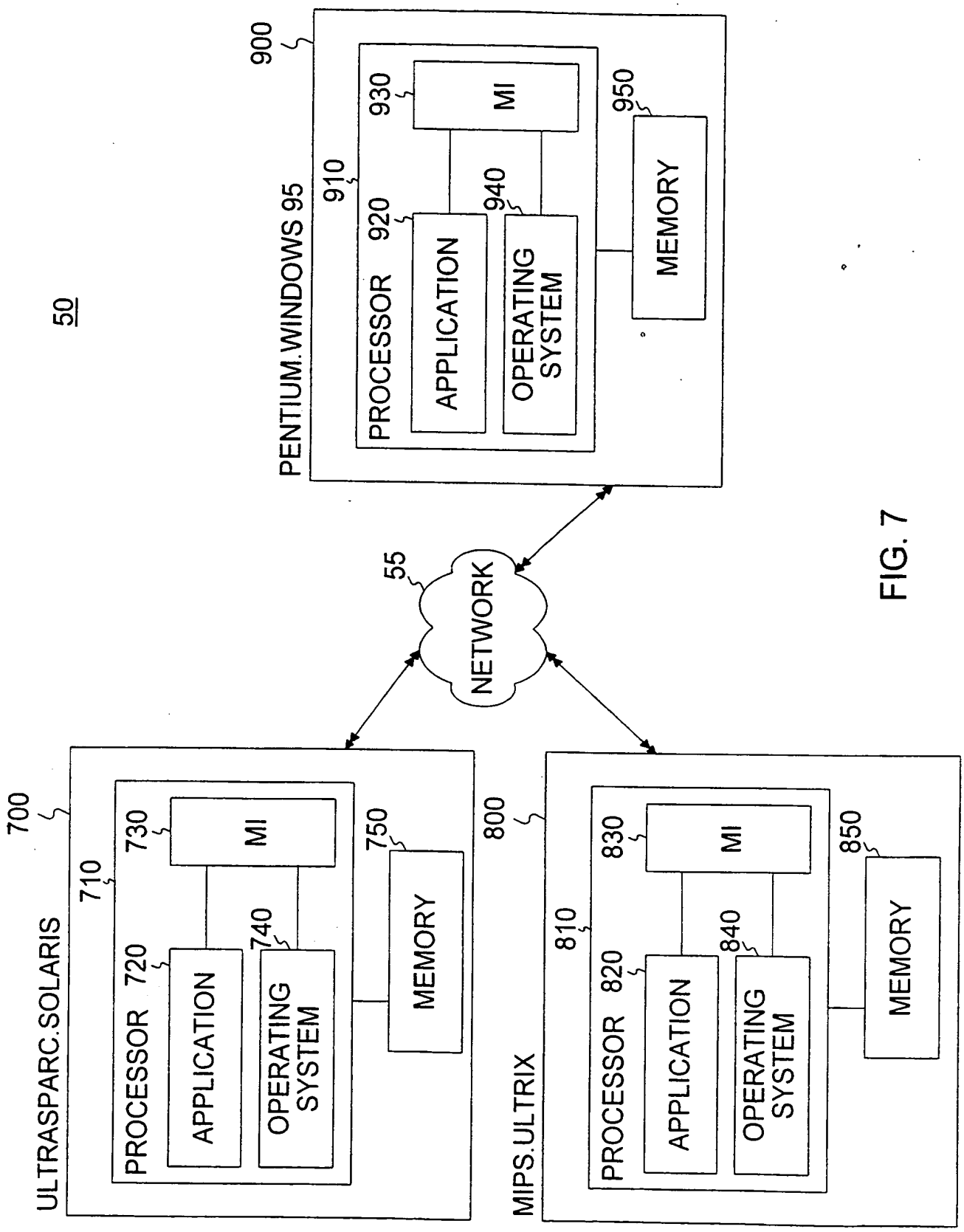


FIG. 7

BNSDOCID: <WO___9944129A1_I_>

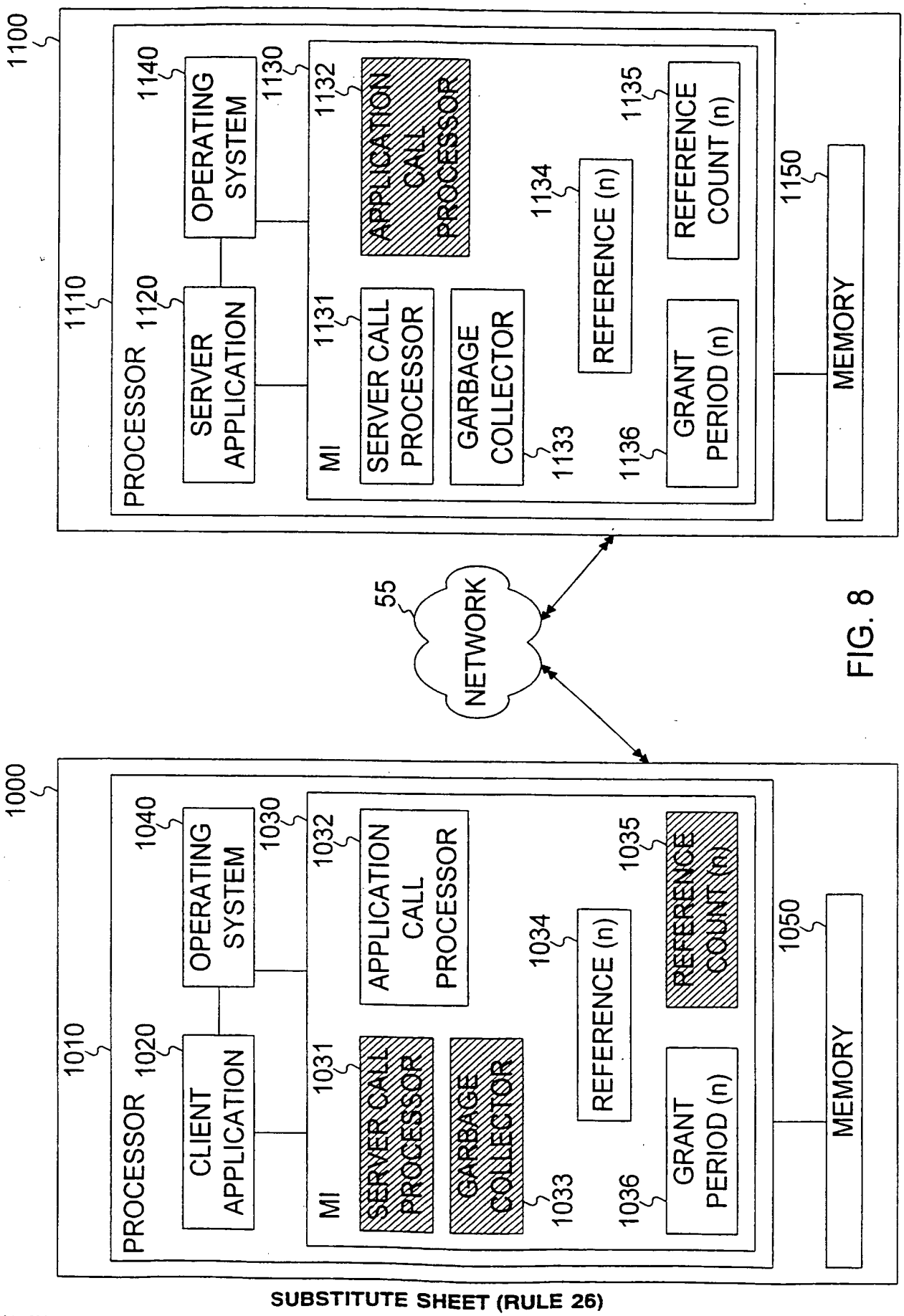

1100
1110
PROCESSOR
1120
SERVER APPLICATION
1140
OPERATING SYSTEM
1130
MI
1131
SERVER CALL PROCESSOR
GARBAGE COLLECTOR
1133
1132
APPLICATION CALL PROCESSOR
1134
REFERENCE (n)
1135
REFERENCE COUNT (n)
1136
GRANT PERIOD (n)
1150
MEMORY
55
NETWORK
1000
1010
PROCESSOR
1020
CLIENT APPLICATION
1040
OPERATING SYSTEM
1030
MI
1031
SERVER CALL PROCESSOR
GARBAGE COLLECTOR
1033
1032
APPLICATION CALL PROCESSOR
1034
REFERENCE (n)
1035
REFERENCE COUNT (n)
1036
GRANT PERIOD (n)
1050
MEMORY

FIG. 8

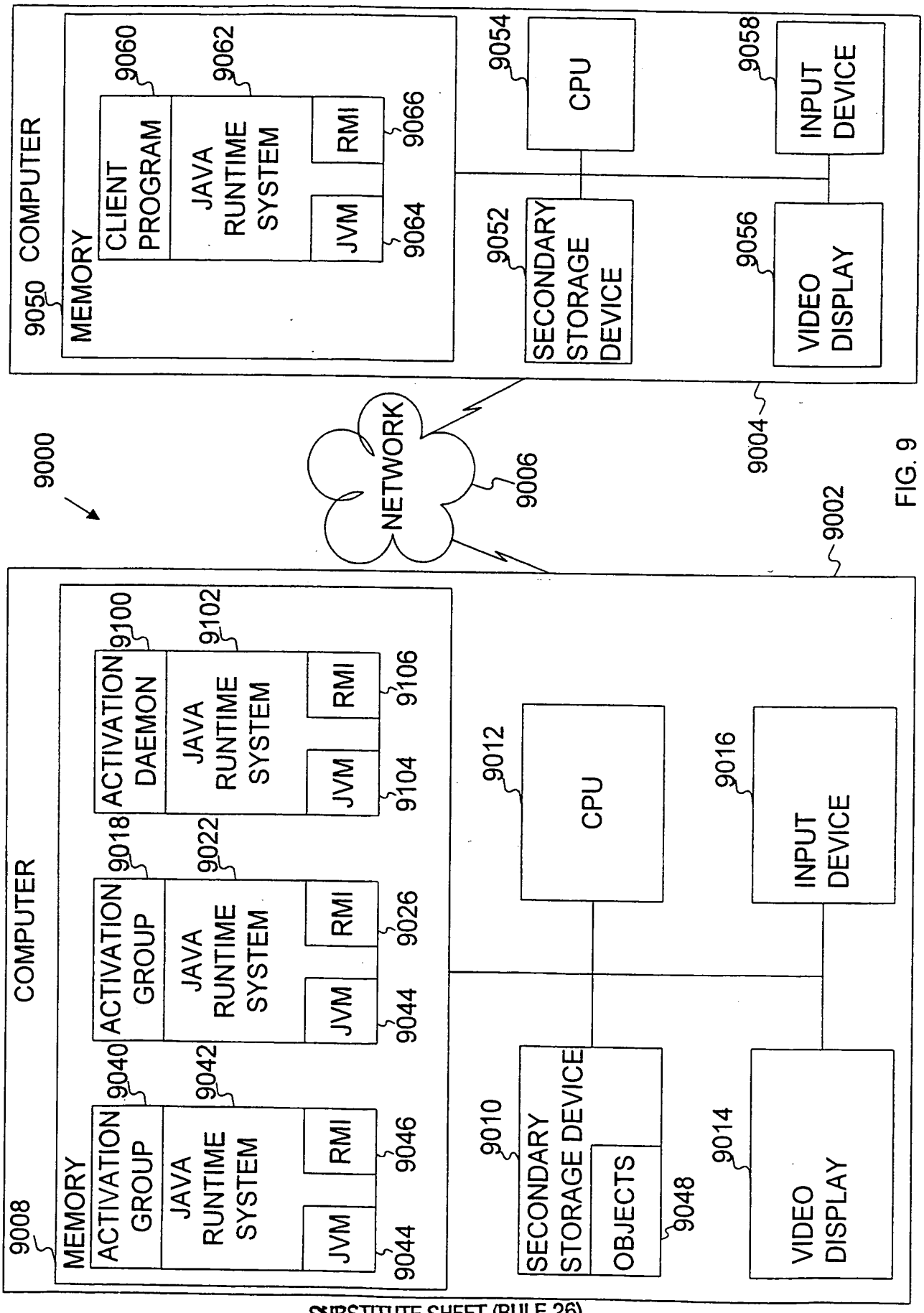


FIG. 9

BNSDOCID: <WO___9944129A1_I_>

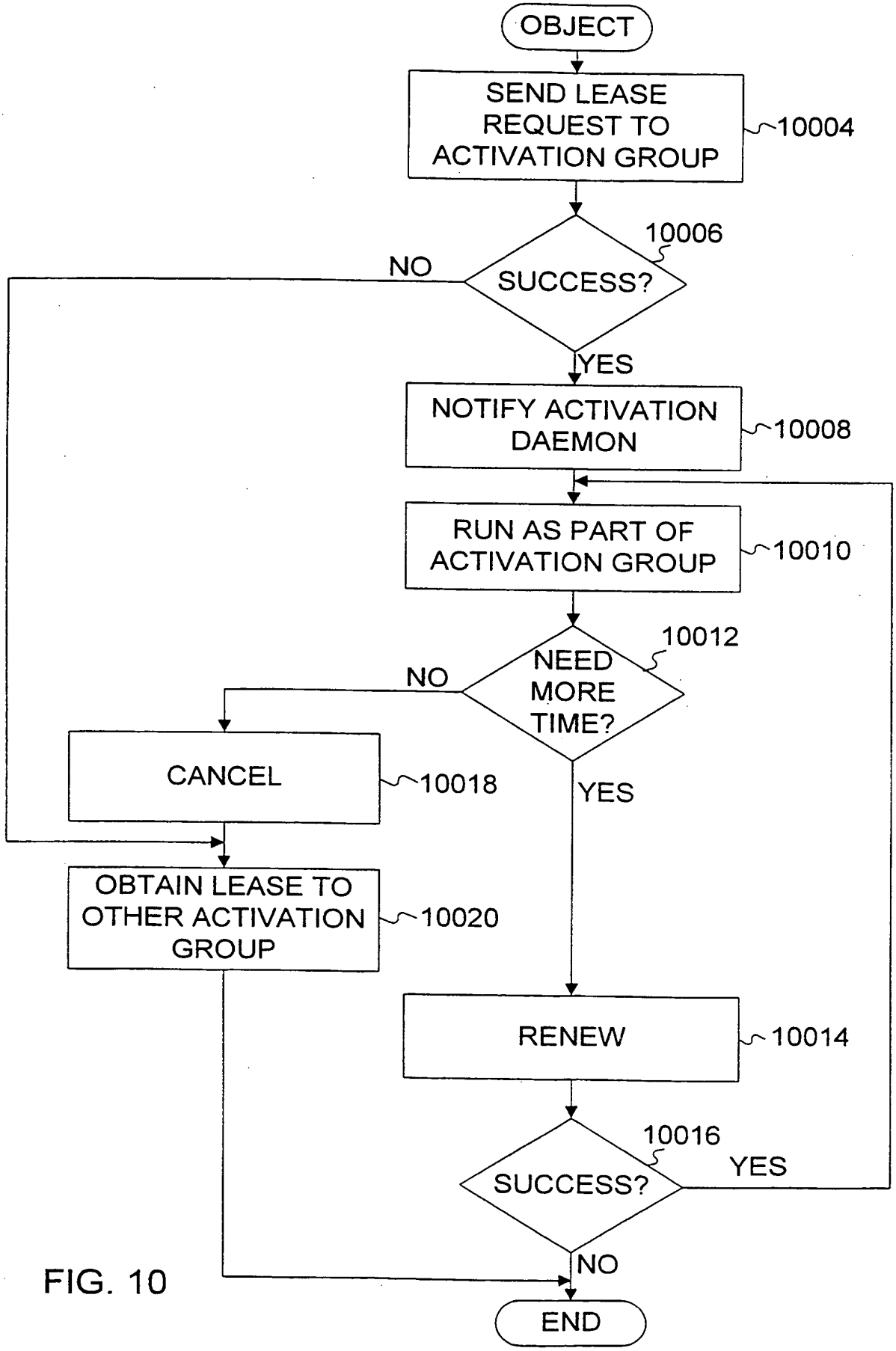


FIG. 10

BNSDOCID: <WO 9944129A1 I >

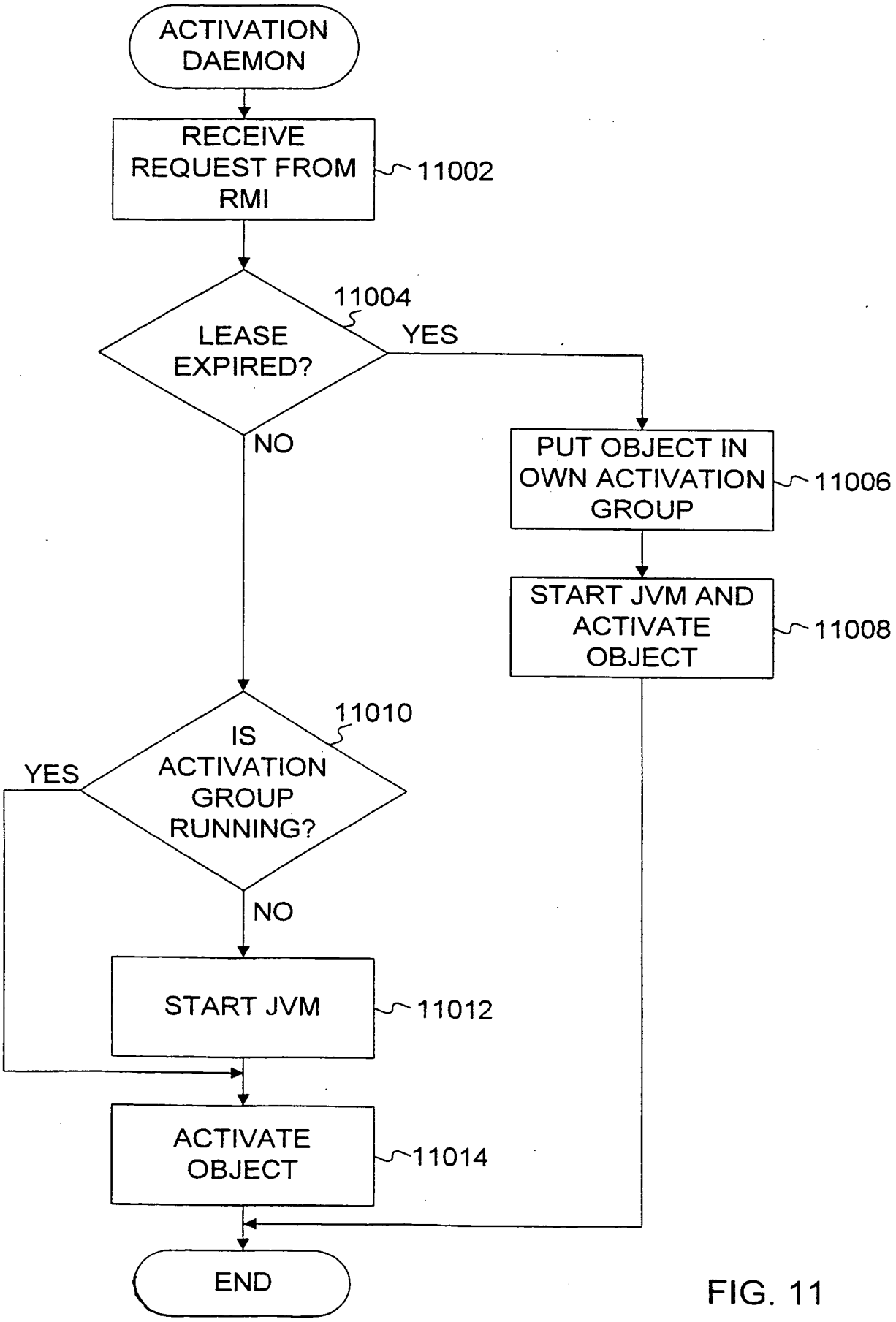


FIG. 11

BNSDOCID: <WO___9944129A1_I_>

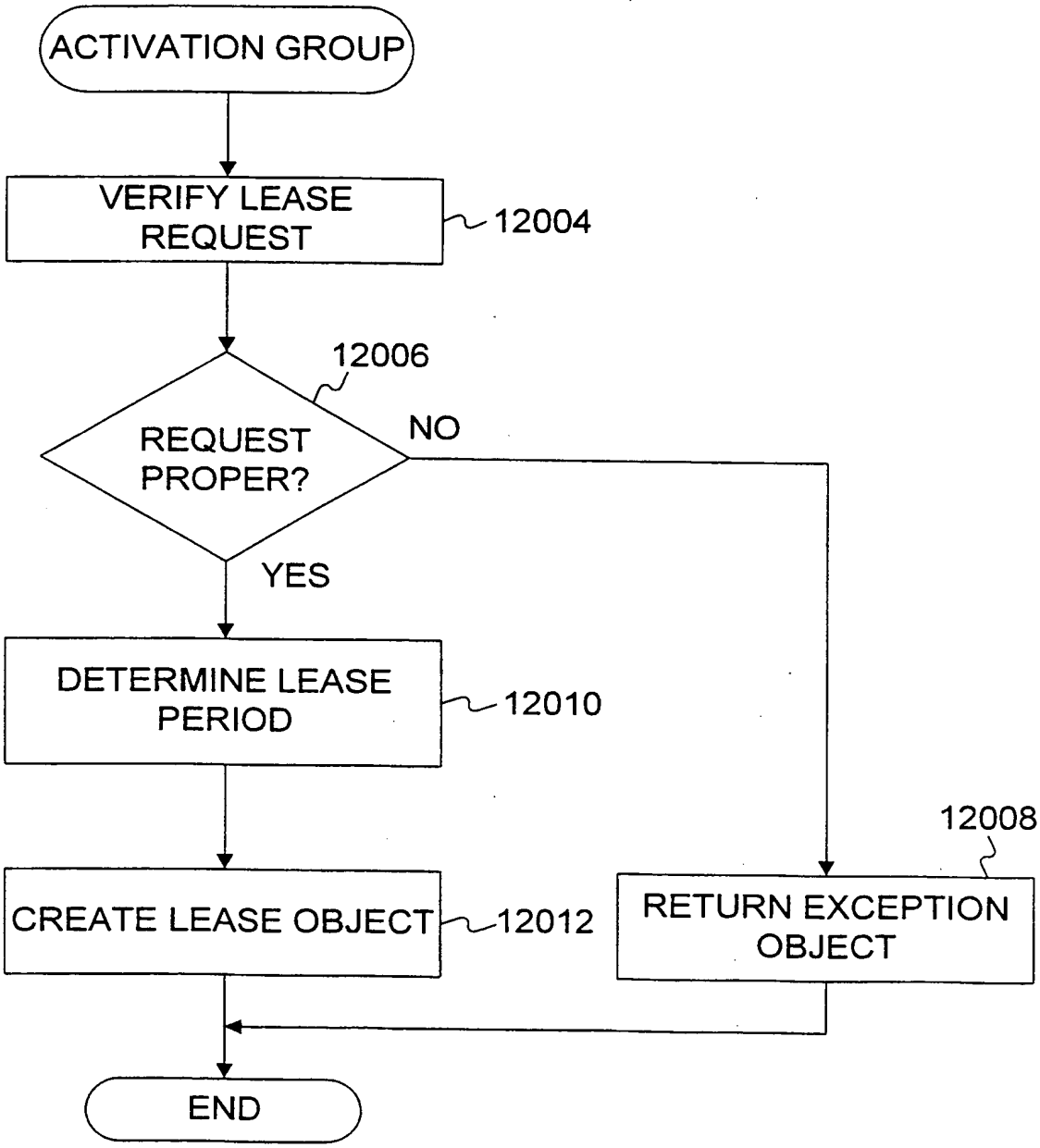

ACTIVATION GROUP
VERIFY LEASE REQUEST
12004
12006
REQUEST PROPER?
NO
YES
DETERMINE LEASE PERIOD
12010
12008
CREATE LEASE OBJECT
12012
RETURN EXCEPTION OBJECT
END

FIG. 12

INTERNATIONAL SEARCH REPORT

Internatio. Application No
PCT/US 99/03399

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4 939 638 A (STEPHENSON R ASHLEY ET AL) 3 July 1990 (1990-07-03) column 1, line 35 - line 68	1
A	EP 0 569 195 A (ROCKWELL INTERNATIONAL CORP) 10 November 1993 (1993-11-10) column 3, line 21 - column 4, line 1	1
A	ANONYMOUS: "Resource Preemption for Priority Scheduling. November 1973." IBM TECHNICAL DISCLOSURE BULLETIN, vol. 16, no. 6, page 1931 XP002109435 New York, US the whole document	1

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 July 1999	29/07/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Brandt, J

Form PCT/ISA/210 (second sheet) (July 1992)

BNSDOCID: <WO___9944129A1_I_>

INTERNATIONAL SEARCH REPORT

Information on patent family members

Internatio. Application No
PCT/US 99/03399

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 4939638 A	03-07-1990	NONE	
EP 0569195 A	10-11-1993	US 5353343 A	04-10-1994
		CA 2095191 A	31-10-1993
		JP 6284458 A	07-10-1994

BNSDOCID: <WO___9944129A1_I_>